Dynamic Graph Sampling Service for Real-time GNN Inference at Scale

Jie Sun[†] Li Su[‡] Wenting Shen[‡] Zichao Zhang[‡] Zuocheng Shi[†] Jingbo Xu[‡]
Yong Li[‡] Wenyuan Yu[‡] Zeke Wang[†] Fei Wu[†] Jingren Zhou[‡]
Zhejiang University[†]   Alibaba Group[‡]

Background

GNN Inference Service on Dynamic Graph

In industrial GNN scenarios, sampling-based mini-batch training is widely adopted to scale out GNN applications to very large graphs, where receptive fields for vertices are obtained via neighbor sampling during both model training and inference. Since the structure and attributes of real-world graphs change continuously, it is imperative that the inferred vertex representation can accurately reflect these dynamic updates in real-time. GNN inference services on dynamic graph mush achieve latency-related SLOs. For example, GNN-based real-time recommendation systems often require stable millisecond-level latency performance.

Challenge

(1) High time complexity of K-hop sampling;
(2) I/O overheads of disk-based/distributed graph storage;
(3) Imbalanced workload due to graph skewness.

System Overview

Graph Update Flow   Subscription Event   Pre-sampling Data Flow
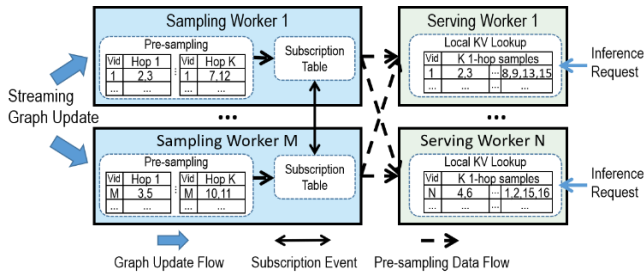
Target:
Real-time GNN inference service on large-scale dynamic graph.

Insights:
GNN inference service is query-aware.

Architecture:
DGS decouples graph sampling and GNN inference physically.
Sampling worker: Each sampling worker is responsible for a specific graph partition: conducting pre-sampling for the decomposed one-hop sampling queries and delivering the sampling results to the serving workers.
Serving worker: Each serving worker caches the sampling results of K one-hop queries received from sampling workers and serves the inference requests for a partition of vertices in the graph.

Key Designs

Sampling Worker

Step 1: Decompose K-hop query into K 1-hop queries
Example: 2-hop Query
```
g.V("user", feed_id).alias("seed")
  .OutV("click").sample(2).by("random")
  .OutV("swing").sample(2).by("random").values
```
Decomposed into:
1-hop query Q_1:
```
V.OutV("click").sample(2).by("random")
```
1-hop query Q_2:
```
V.OutV("swing").sample(2).by("random")
```
Exclusively store the 1 hop samples on sampling workers

Step 2: Event-driven reservoir sample
Graph update event: Edge_Type (src_v, dst_v, timestamp)
E.g., `click(1, 5, `T_1`); swing(4, 8, `T_2`)`
Reservoir sampling: Randomly replace existing samples

Time	Vid	Q_1	Vid	Q_2	Updated?
T_0	1	2, 3	4	6, 7	/
T_1	1	2, 5	4	6, 7	yes
T_2	1	2, 5	4	6, 7	no

Step 3: Update the subscription table

Time	Hop	Vid	Subscribed by
T_0	Q_1	1	Serving worker 1
T_0	Q_2	3	Serving worker 1
T_1	Q_1	1	Serving worker 1
T_1	Q_2	5	Serving worker 1

Serving Worker
Handle inference request for a partition of vertices
Generate K-hop samples by local KV Lookups.

Worker	Vid	Q_1	Q_2
1	1	2, 5	8, 9, 10, 13

Worker	Vid	Q_1	Q_2
2	8	4, 9	6, 7, 15, 16

Evaluation

Settings:
(1) Real Alibaba e-commerce datasets, #V=240M, #E=6.1B;
(2) 2-hop random sampling query, fan-outs=[15,10];
(3) Single machine with 64 * Intel(R) Xeon(R) CPU E5-2682 v4, 256GB host memory, 960GB SAMSUNG NVMe SSD

Serving Latency	P99: 20ms
Serving Throughput	20,000 QPS
Update Ingestion Throughput	109MB/s

Serving & update ingestion throughput can scale out linearly.

DGS is open-sourced as a part of Graph-Learn:
https://github.com/alibaba/graph-learn